Tarea 3 - JavaUNO-GUI

Profesor:	Alexandre Bergel
Auxiliares:	Javier Espinoza
	Eduardo Riveros
Ayudantes	Marco Caballero
	Franco Cruces
	Matilde Rivas
	Juan-Pablo Silva
Fecha de Entrega:	28 de Noviembre del 2017

1. El Juego

UNO es un juego de cartas para dos o más jugadores, jugado con un mazo de cartas no convencional, en el que el objetivo del jugador es quedarse sin cartas en la mano antes que sus rivales, siguiendo un conjunto de reglas de fácil aprendizaje.

2. La Tarea

El objetivo de esta tarea consiste en extender el código solicitado en la tarea 2 con nuevas reglas, nuevos jugadores automáticos y una interfaz gráfica más amigable.

2.1. Nuevas reglas y mecánicas de juego

Para hacer el juego más dinámico, modificarán algunas reglas de la implementación anterior:

1. Cadenas de robo de cartas: Si el *Draw Well* no está vacío y el jugador actual tiene una carta para robar en su mano. Esta carta debe ser jugada automáticamente en su turno para evitar ser afectado por la cadena de robos.
2. Robar para siempre: Si un jugador no tiene ninguna carta que pueda jugar, debe robar del mazo hasta encontrar una que sea jugable. Si el jugador tiene cartas para jugar, no puede robar del mazo.
3. Cartas Especiales de Color y Comodines: Es necesario crear una nueva carta especial de color (con una habilidad decidida por ustedes) y nueva una carta comodín (con una habilidad decidida por ustedes) al mazo de cartas regular.
    - Las habilidades que decidan no pueden ser del tipo “robar n cartas” o “cambiar de color”.
    - Se les otorgará una carta con dibujos de estrella para ambas mecánicas, en caso que les complique hacer un dibujo personalizado.

2.2. Interfaz Gráfica Amigable

En esta tarea además tendrán que rehacer los paquetes "controller" y "view", utilizando el framework *JavaFX* para la nueva interfaz.

Para guiarlos en la construcción de la vista, les adjuntamos algunos *mockup* sugeridos con la información mínima que debería mostrarse al usuario en el juego.

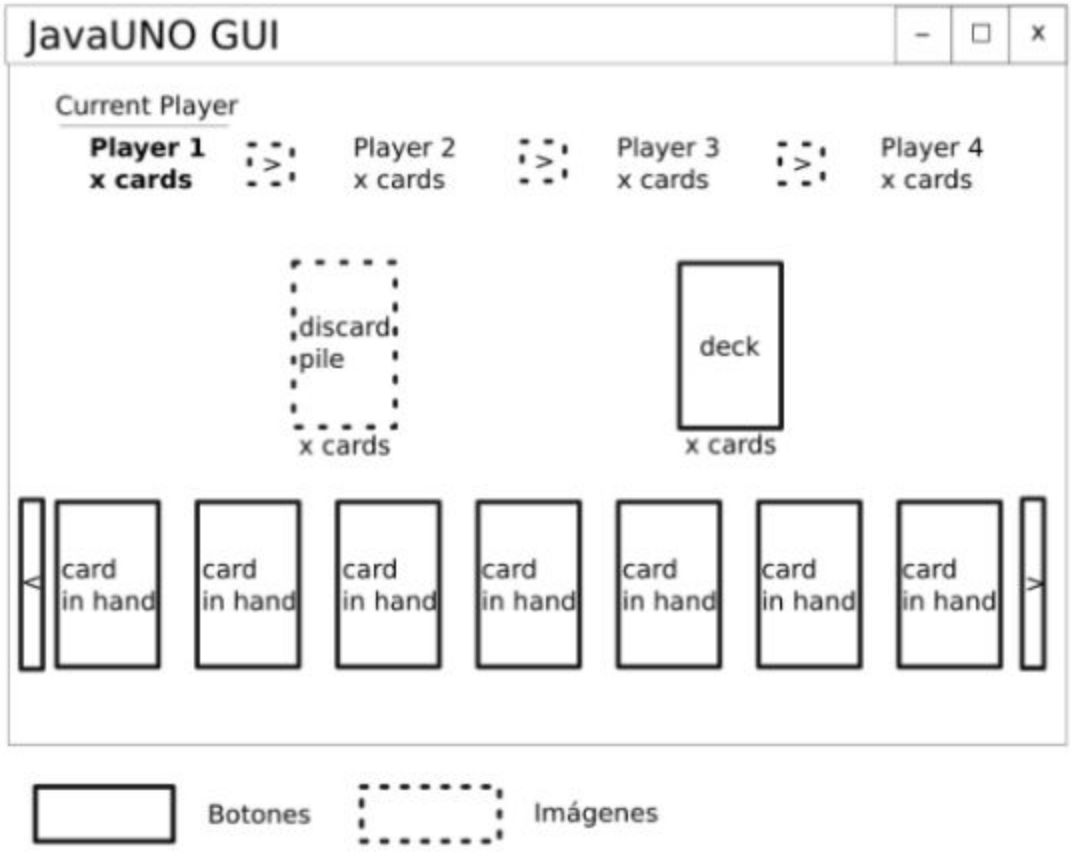

Figura 1.- Mockup de presentación del juego JavaUNO GUI

En la figura anterior se puede observar las siguientes zonas del juego:

- La fila superior contiene el nombre y las cartas en mano del jugador actual y los 3 próximos jugadores. Entre cada jugador hay un dibujo de una flechita, indicando la secuencia.
    - **Esta información debe actualizarse en tiempo real cada vez que cambia debido al uso de alguna carta.**
- La fila intermedia posee una imagen de la carta actual en la pila de descartes, y un botón del mazo. Para robar cartas del mazo, basta con hacer click en este botón.
- La fila inferior posee todas las cartas que el jugador humano actual puede jugar.

- Las cartas se muestran de a 7. Si el jugador tiene más cartas, puede verlas usando los botones de paginado al lado de ellas (Representados por botones alargados con flechitas en la imagen)
- Si el jugador actual es un jugador virtual, no muestra ninguna carta (Todos los botones están deshabilitados).

Dado que es necesario mostrar cierta información extra al jugador a medida transcurre el juego, es de utilidad contar con un sistema de ventanas emergentes como el de la Figura 2:

Figura 2.- Ejemplos de ventanas emergentes para el juego

Tal como mencionan las imágenes, es necesario contar con ventanas emergentes para, al menos, las siguientes 2 mecánicas:

- Dar a conocer errores e informaciones al jugador (Carta no válida, jugador ganador, etc).
- Solicitar información extra al jugador (Color al jugar comodín, confirmar reinicio de partida o algún requisito impuesto por sus cartas estrella).

El uso de los mockup anterior no es obligatorio. Si se les ocurre una forma más atractiva, cómoda e intuitiva de mostrar la misma información como mínimo, están invitados a usarla, siempre y cuando la justifiquen correctamente en su informe.

Para toda esta sección (En especial para entender y conocer las API de los componentes de JavaFX) les recomendamos utilizar como referencia el tutorial de TutorialsPoint de JavaFX

3. Flujo de Juego

Para facilitar la comprensión del código entregado, a continuación se explica el flujo de juego de una partida de UNO según el código desarrollado en la Tarea 2.

Para iniciar cada turno:

- Revisar si el jugador actual tiene que robar cartas.
  - Si tiene que robar cartas y tiene una carta para alargar la cadena, tirarla automáticamente y terminar el turno.
  - Si no, robar las cartas y terminar el turno.
- Solicitar al jugador que tire una carta:
  - Si el jugador no tiene cartas que sean jugables, deberá poder robar del mazo hasta que le salga una jugable. Al momento de tener cartas jugables, se deshabilita el robo desde el mazo.
- El jugador podrá hacer click en una carta para jugarla
  - Si la carta es jugable, la jugada se hará efectiva.
    - Si es una carta comodín, se abrirá una ventana solicitando el color a seleccionar (Como en la figura 2).
    - Si la carta tiene una acción a ejecutar, se ejecuta ésta, cambiando el estado del juego.
  - Si la carta no es jugable, se mostrará una ventana informativa alertando de la situación.
- Luego de jugar la carta
  - Si es necesario, gritar "UNO!" automáticamente por el jugador, e informarlo a través de una ventana informativa.
  - Revisar si el jugador cumple las condiciones de término de juego. De ser así, ´se informa mediante una ventana de información quién es el ganador, cerrándose el juego al cerrar esa ventana.

4. Requisitos Tarea 3

Para la tarea 3, deberán implementar **todo lo mencionado en el enunciado hasta ahora** sobre la Tarea 2**:**

- Interfaz Gráfica Amigable
- 4 nuevas mecánicas de juego.

Por defecto, todos trabajarán con un **código de base** entregado por el equipo docente y desarrollado por uno de sus compañeros en la iteración anterior. En caso que deseen usar su tarea 2 como base de la tarea 3, deberán dar aviso al equipo docente para tener constancia del hecho. Los alumnos que usen su propio código de la tarea 2 como base contarán con **0.5 décimas de bonus** para usarse al final del cálculo de la nota de la tarea 3.

5. Requerimientos Adicionales

Además de una implementación basada en las buenas prácticas y patrones de diseño vistos en clases, usted debe considerar el cumplimiento de los puntos siguientes:

- **Cobertura**: Cree los test unitarios, en JUnit, que sean necesarios para alcanzar un **90%** de coverage por paquete. Estos tests deben estar en el paquete *test* de su proyecto.
- **Javadoc:** Cada clase y método público debe estar debidamente documentado, siguiendo las convenciones de Javadoc.
- **Resumen y descripción de su implementación:** Debe entregar un archivo en formato *.pdf* que contenga lo siguiente:
    a. **Portada:** debe incluir el nombre de la tarea, los datos del curso, su nombre y un link al repositorio en que trabajó.
    b. **Patrones de diseño**: Una sección en la que identifica y comenta los patrones de diseño en la tarea (Tanto los agregados por usted como los que venían en el código de la tarea 2)
    c. **Cartas Extra:** Una sección explicando lo que hacen las cartas extra de la tarea.
    d. **UML:** Un diagrama UML por paquete del proyecto (excluyendo *tests*).
    e. **Otros:** Además, debe incluir todas las justificaciones que encuentren pertinentes sobre su implementación para hacerla más entendible al ayudante que revise.

  No existe extensión mínima para b, c ni e, por lo que valoramos que sean concisos con la información entregada, procurando que sirva para dar a entender mejor su trabajo.
- **Coding Style:** El código y la documentación debe seguir las convenciones de Google.
- **GitHub:** Debe mantener actualizado su repositorio de GitHub, ya que de él se extraerá la tarea que será revisada.

6. Evaluación

- **Funcionalidad (Descuento máximo de 1,5 punto):** Corresponde al rendimiento en tests que demuestran que la tarea hace lo que se pide en este enunciado.
- **Diseño (Descuento máximo de 1,5 puntos)**: Corresponde al uso de patrones de diseño y materia vista en clases de forma adecuada a cada desafío presentado por la tarea.
- **Coverage (Descuento máximo de 1 punto)** correspondiente a cubrir con tests hechos por ustedes mismos, al menos **el 90% del código del paquete** *model.* (Los paquetes controller y view siguen sin necesidad de Coverage).
- **Resumen (Descuento máximo de 1 punto)** cumpliendo lo mencionado en el punto *Requerimientos Adicionales*.
- **Code Smells (Descuento máximo de 1 punto)** correspondiente a no presentar los *code smells* vistos en clases, usar *Javadoc* donde corresponda y a utilizar el *code style* solicitado.

Recuerden que no es suficiente para una buena evaluación que la tarea funcione, lo más importante es que estén aplicando los conocimientos vistos en clases.

7. Entrega

1. La implementación de su tarea se entregará **solamente por GitHub**, correspondiendo a la entrega de la carpeta raíz del proyecto en Eclipse (no olviden agregar el archivo *.gitignore* de Material Docente a su proyecto)
2. El Resumen se entregará **solamente por U-Cursos**.
3. Tanto la implementación como el informe tienen de plazo de entrega máximo el estipulado en la tarea de U-Cursos.
4. El repositorio **privado** de su tarea debe llamarse **cc3002-tarea3.** De esta forma, si su usuario en Github es *adderou*, su tarea se ubicará en *https://github.com/adderou/cc3002-tarea3*
5. Recuerden invitar a la cuenta *CC3002EquipoDocente* como integrante del repositorio para poder revisar la tarea.
6. **Solo podrán realizar commits hasta la hora límite de entrega de la tarea en U-Cursos.** En caso que detectemos commits en horas posteriores, se les evaluará con nota mínima.

8. Recomendaciones Finales

- Les recomendamos que **partan por la creación de la GUI para el juego,** debido a que sus funcionalidades ponderarán más en la nota final.
- Se les solicita que todas las consultas referentes a la tarea las hagan en primera instancia a través del **foro de U-Cursos**. En caso que nos demoremos mucho en contestar (1 día hábil o más), pueden hacernos llegar un correo a los auxiliares o ayudantes para avisarnos de sus consultas.